QBE: QLearning-Based Exploration of Android Applications
Presenter: Yavuz Koroglu

Yavuz Koroglu and Alper Sen

Ozlem Muslu, Ceyda Ulker, Yunus Mete, Tolga Tanriverdi, and Yunus Donmez

Dependable Systems Group (DSG)
Bogazici University, Istanbul, Turkey
http://depend.cmpe.boun.edu.tr
{yavuz.koroglu,alper.sen}@boun.edu.tr

NETAS Telecommunications
Istanbul, Turkey
yunusm@netas.com.tr

11th IEEE Conference on Software Testing, Validation and Verification (ICST’18)
Overview

1. Introduction
2. A Real Crash Example
3. QLearning-Based Exploration (QBE)
4. An Illustrative Example of QLearning
5. Evaluation
6. Conclusions and Future Work
Motivation

Mobile GUI Applications are Ubiquitous

- We use mobile phones often (3 hours/day)
- Mostly on mobile applications (90% of the time spent)

Android Market is Growing

- 2.6 billion mobile phone users

Android has the Largest Share

- 82.8% of all apps are for Android
Publicly Available Automated Android GUI Testing Tools

- Monkey
- A$^3$E
- SwiftHand
- PUMA
- DynoDroid
- Sapienz

**Monkey**

Outperforms other tools in terms of

- Coverage
- Crashes
Monkey

- Developed by Google
- Generates random
  1. System events and
  2. GUI actions
- Built-in (comes with the Android OS)
# Pros/Cons of Monkey

## Advantages

- **High Variety of Events**  
  (Sensor, Navigation, System Events, Basic Gestures)
- **High Event Rate**  
  (thousands of events per second)

## Disadvantages

- **Reproducibility Issues** (Poor Verifiability)
- **Misses Deep Crashes and Deep Activities**
A Real Crash Found by None of the Other Tools

→ A GPS application.
→ Previous Actions: (1) **reinitialize**
→ Next Action: **menu**
A GPS application.

Previous Actions:
(1) reinitialize, (2) menu

Next Action: click More
A Real Crash Found by None of the Other Tools

→ A GPS application.
→ Previous Actions:
(1) reinitialize, (2) menu, (3) click More
→ Next Action: click show arrow view
A Real Crash Found by None of the Other Tools

→ A GPS application.
→ Previous Actions:
  (1) reinitialize, (2) menu, (3) click More, (4) click show arrow view
→ Next Action: menu
A Real Crash Found by None of the Other Tools

→ A GPS application.
→ Previous Actions:
  (1) reinitialize, (2) menu, (3) click More, (4) click show arrow view, (5) menu
→ Next Action: click cache view
A Real Crash Found by None of the Other Tools

→ A GPS application.
→ Previous Actions:
   (1) reinitialize, (2) menu, (3) click More, (4) click show arrow view,
   (5) menu, (6) click cache view
→ CRASH
→ Monkey: Probability of generating these actions in this order is very low.
→ Others: It takes a long time to systematically exhaust all possibilities.
QLearning-Based Exploration (QBE) Overview

Main Idea

- To **learn** the best actions in similar states.

Main Flow

1. **Explore** the training set (with random exploration)
2. **Generate** GUI Models
3. **Learn** the best transitions
4. **Direct** the testing process (use the learned model)

Figure: QLearning-Based Exploration (QBE) Overview
In general,

- Most applications do **NOT** have a model
- Learn the application model **dynamically**
- The model is an **Extended Labeled Transition System (ELTS)** where
  1. **Nodes** are GUI **states**.
  2. **Edges** are transitions via GUI **actions**.
GUI State

1. Java Package Name
2. Activity Name
   (An activity roughly corresponds to an Android screen)
3. Contextual Attributes
   (WiFi, Orientation etc.)
4. GUI Components (widgets)
   on the screen
User-triggered events: **text, click, swipe** etc.
AndroFrame: Automated Test Generation Framework

What is AndroFrame?

It is a
- Fully-automated,
- Black-box,
- Modular,
- Automata Learning
replayable test case generation framework.

Important

- We build QBE on top of AndroFrame.

Figure: Example Model of the Yahtzee App
Main Idea

- QLearner observes
  1. The current **state** and
  2. The latest **reward**
- QLearner decides on
  1. An action
QLearner

Main Idea

- QLearner observes
  1. The current state and
  2. The latest reward
- QLearner decides on
  1. An action

Q-Matrix

A matrix of values where
- Rows are states and
- Columns are actions.

Q-Value

- Cells in the Q-Matrix.
- Associated with a state-action pair.
- Expectancy of the action getting a reward in the next state.
Main Idea

- QLearner observes
  1. The current **state** and
  2. The latest **reward**
- QLearner decides on
  1. An action

Example

<table>
<thead>
<tr>
<th></th>
<th>click</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s3</td>
<td>0.17</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Important

- All rows add up to 1 (except unvisited states)
- At **s1**, always **click**
- At **s2**, no knowledge (all 0s)
- At **s3**, mostly **text**
**Main Idea**

- QLearner observes
  1. The current **state** and
  2. The latest **reward**
- QLearner decides on
  1. An action

Initially, $\tilde{Q} = 0$.

Flowchart:
- **Choose Action from $\tilde{Q}$**
- **Perform Action**
- **Measure Reward**
- **Update $\tilde{Q}$**

**Initial State**
- QLearner observes the current state and the latest reward.

**Decision Process**
- QLearner decides on an action.

**Update Process**
- The Q-value function $\tilde{Q}$ is updated based on the chosen action and its reward.

**Loop**
- The process repeats with the updated $\tilde{Q}$, leading to a decision on another action and so on.
QLearning: Standard Updates

$$
\begin{aligned}
\tilde{Q}[s, a] &\leftarrow \tilde{Q}[s, a] + N[s, a]^{-1} \\
\text{Next Q-Matrix} &+ \text{Previous Q-Matrix} + \text{History Matrix} \\
& \left[ o(v, z) + \gamma \tilde{Q}[s', a'] - \tilde{Q}[s, a] \right]
\end{aligned}
$$

**Definitions**

- **History Matrix:** A running count of previous updates on each $Q[s, a]$.
- **Objective Function:** Denotes the reward. 1 if the goal is satisfied, 0 otherwise.
- **Future Expectancy:** Allows future rewards to be propagated along an execution path.
- **Discount Factor ($\gamma$):** A value btw 0 and 1 to decrease the future expectancy as the path gets longer.
Illustrative Example: How QLearning Works

Without Abstraction

- 7 application states (excluding "_" and "CRASH")
- 11 state-action pairs (excluding "reinitialize")
- Would be too large in real scenarios.

Similar States

- Cosine Similarity > 0.95
  1. v1, v1’, v1”, v1’’
  2. v2, v2’, v2”

Figure: GUI Model of the Yahtzee App
Illustrative Example: How QLearning Works

Figure: GUI Model of the Yahtzee App

Let’s Abstract

- States (2 state types)
  1. $s_1 = \{v_1, v_1', v_1'', v_1'''\}$
  2. $s_2 = \{v_2, v_2', v_2''\}$

- Actions (2 action types)
  1. click
  2. text

- We get a 2 by 2 matrix: $\tilde{Q}[s, a]$
Illustrative Example: How QLearning Works

The only way to update Q-values is to
- Get a reward

Initial Q-Matrix

<table>
<thead>
<tr>
<th>State</th>
<th>Click</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure: GUI Model of the Yahtzee App
Illustrative Example: How QLearning Works

New Q-Matrix

<table>
<thead>
<tr>
<th></th>
<th>click</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Test Case: $v_1, v_2, v_1, v_1'', v_2''$

- No rewards, no updates.

Figure: GUI Model of the Yahtzee App
Illustrative Example: How QLearning Works

**Figure:** GUI Model of the Yahtzee App

New Q-Matrix

<table>
<thead>
<tr>
<th></th>
<th>click</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Test Case: $v_1, v_1'', v_1''', CRASH$

- Learns the last transition first.
Illustrative Example: How QLearning Works

New Q-Matrix

<table>
<thead>
<tr>
<th></th>
<th>click</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>.53</td>
<td>.47</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Test Case: \(v_1, v_1'', v_1''', \text{CRASH} \) (again)

- Now, \(v_1'' \rightarrow v_1'''\) also gets Q-value, due to future value.

Figure: GUI Model of the Yahtzee App
Illustrative Example: How QLearning Works

Figure: GUI Model of the Yahtzee App

<table>
<thead>
<tr>
<th></th>
<th>click</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>.57</td>
<td>.43</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- At all s2 states (v2, v2', v2''), QBE always **clicks**.
Reward (Objective) Function

Two reward functions
\((v: \text{Current State}, z: \text{GUI Action}, v': \text{Next State})\)

1. Crash Detection

\[
o(v, z) = \begin{cases} 
1 & \text{if } v' \text{ is a CRASH state} \\
0 & \text{otherwise} 
\end{cases}
\]  (1)

2. Activity Coverage Increase

\[
o(v, z) = \begin{cases} 
1 & \text{if } v' \text{ belongs to a new Activity} \\
0 & \text{otherwise} 
\end{cases}
\]  (2)
Common Evaluation Criteria

Number of Distinct Crashes

- **Parse** the Android logs (Common technique)
- **Stack traces for exceptions** are also in these logs
- Do NOT count the same stack trace more than once

Activity Coverage

- A **high level metric** that is necessary to claim a high coverage of functionality (\# Explored Activities / \# All Activities)

Instruction Coverage

- A **low level metric** that shows the amount of code utilization (\# Explored Instructions / \# All Instructions)
Experimental Setup

- 14 x Android-x86 VirtualBox guests (with Android 4.4.r5)
- 300 Android applications randomly selected from F-Droid benchmarks
  - 200 training and 100 test applications
- 10 minutes for each application.
- Implemented 4 Strategies in AndroFrame,
  1. Random Exploration (RE)
  2. Depth-First Exploration (DFE)
  3. Activity-Based QBE (QBEa)
     - Reward function is Activity Coverage Increase.
  4. Crash-Based QBE (QBEc)
     - Reward function is Crash Detection.
Experimental Results

Table: Experimental Results over 10 minutes

<table>
<thead>
<tr>
<th>Tool</th>
<th>Activity (%)</th>
<th>Instr. (%)</th>
<th>#Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AndroFrame</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity-Based QBE (QBEa)</td>
<td>78</td>
<td>40</td>
<td>7.8</td>
</tr>
<tr>
<td>Crash-Based QBE (QBEc)</td>
<td>65</td>
<td>32</td>
<td>12.6</td>
</tr>
<tr>
<td>Depth-First Exploration (DFE)</td>
<td>63</td>
<td>34</td>
<td>3</td>
</tr>
<tr>
<td>Random Exploration (RE)</td>
<td>58</td>
<td>30</td>
<td>3.2</td>
</tr>
<tr>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynoDroid</td>
<td>50</td>
<td>35</td>
<td>5.2</td>
</tr>
<tr>
<td>A³E</td>
<td>41</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>Monkey</td>
<td>60</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>PUMA</td>
<td>64</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>Sapienz</td>
<td>76</td>
<td>44</td>
<td>4</td>
</tr>
<tr>
<td>SwiftHand</td>
<td>40</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

QBEa has the best activity coverage.


### Experimental Results

**Table: Experimental Results over 10 minutes**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Activity (%)</th>
<th>Instr. (%)</th>
<th>#Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AndroFrame</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity-Based QBE (QBEa)</td>
<td>78</td>
<td>40</td>
<td>7.8</td>
</tr>
<tr>
<td>Crash-Based QBE (QBEc)</td>
<td>65</td>
<td>32</td>
<td>12.6</td>
</tr>
<tr>
<td>Depth-First Exploration (DFE)</td>
<td>63</td>
<td>34</td>
<td>3</td>
</tr>
<tr>
<td>Random Exploration (RE)</td>
<td>58</td>
<td>30</td>
<td>3.2</td>
</tr>
<tr>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynoDroid</td>
<td>50</td>
<td>35</td>
<td>5.2</td>
</tr>
<tr>
<td>A³E</td>
<td>41</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>Monkey</td>
<td>60</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>PUMA</td>
<td>64</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>Sapienz</td>
<td>76</td>
<td>44</td>
<td>4</td>
</tr>
<tr>
<td>SwiftHand</td>
<td>40</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

*Sapienz* has better code coverage.
Experimental Results

Table: Experimental Results over 10 minutes

<table>
<thead>
<tr>
<th>Tool</th>
<th>Activity (%)</th>
<th>Instr. (%)</th>
<th>#Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AndroFrame</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity-Based QBE (QBEa)</td>
<td>78</td>
<td>40</td>
<td>7.8</td>
</tr>
<tr>
<td>Crash-Based QBE (QBEc)</td>
<td>65</td>
<td>32</td>
<td>12.6</td>
</tr>
<tr>
<td>Depth-First Exploration (DFE)</td>
<td>63</td>
<td>34</td>
<td>3</td>
</tr>
<tr>
<td>Random Exploration (RE)</td>
<td>58</td>
<td>30</td>
<td>3.2</td>
</tr>
<tr>
<td>Others</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynoDroid</td>
<td>50</td>
<td>35</td>
<td>5.2</td>
</tr>
<tr>
<td>A$^3$E</td>
<td>41</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>Monkey</td>
<td>60</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>PUMA</td>
<td>64</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>Sapienz</td>
<td>76</td>
<td>44</td>
<td>4</td>
</tr>
<tr>
<td>SwiftHand</td>
<td>40</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

**QBEc** detects the highest number of crashes.
### Experimental Results

<table>
<thead>
<tr>
<th>Tool</th>
<th>Activity (%)</th>
<th>Instr. (%)</th>
<th>#Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AndroFrame</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity-Based QBE (QBEa)</td>
<td>78</td>
<td>40</td>
<td>7.8</td>
</tr>
<tr>
<td>Crash-Based QBE (QBEc)</td>
<td>65</td>
<td>32</td>
<td>12.6</td>
</tr>
<tr>
<td>Depth-First Exploration (DFE)</td>
<td>63</td>
<td>34</td>
<td>3</td>
</tr>
<tr>
<td>Random Exploration (RE)</td>
<td>58</td>
<td>30</td>
<td>3.2</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynoDroid</td>
<td>50</td>
<td>35</td>
<td>5.2</td>
</tr>
<tr>
<td>A³E</td>
<td>41</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>Monkey</td>
<td>60</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>PUMA</td>
<td>64</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>Sapienz</td>
<td>76</td>
<td>44</td>
<td>4</td>
</tr>
<tr>
<td>SwiftHand</td>
<td>40</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

QBE is successful at **coverage** and **crash detection**
Conclusions and Future Work

Conclusions

- QLearning-Based Exploration (QBE) for Model Based GUI Testing of Android Applications
- Experiments on 100 applications. QBE
  1. Achieves the highest activity coverage and
  2. Finds the most distinct crashes.

Future Work

- More reward functions, e.g. code coverage increase.
- Improve abstraction functions.
- Online QLearning for app-specific patterns.
- Use other Machine Learning techniques to improve testing.
An Automatically Generated Test Case

1. Yahtzee
   Amount of Players:
   Amount of Rounds:
   Play

2. Yahtzee
   Amount of Players:
   Amount of Rounds:
   Play

3. Yahtzee
   Amount of Players:
   Amount of Rounds:
   Amount of Rounds and Players must not be zero.
   OK
Mutated Test Case

Yahtzee
Amount of Players:

Yahtzee
Amount of Players:

Unfortunately, Yahtzee has stopped.
Thank You! Any Questions?
Appendix A: Recent Results Across Time

Shows that AndroFrame finds distinct crashes from very early on.
## Appendix B: Table of GUI Actions

### Table: List of GUI Actions for our Automated Testing Tool

<table>
<thead>
<tr>
<th>Non-contextual</th>
<th>Param1</th>
<th>Param2</th>
<th>Param3</th>
<th>Param4</th>
<th>Param5</th>
</tr>
</thead>
<tbody>
<tr>
<td>click</td>
<td>x</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>longclick</td>
<td>x</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>text</td>
<td>x</td>
<td>y</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>swipe</td>
<td>x1</td>
<td>y1</td>
<td>x2</td>
<td>y2</td>
<td>duration</td>
</tr>
<tr>
<td>menu</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>back</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contextual</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectivity</td>
<td>on/off/toggle</td>
</tr>
<tr>
<td>bluetooth</td>
<td>on/off/toggle</td>
</tr>
<tr>
<td>location</td>
<td>gps/gps&amp;network/off/toggle</td>
</tr>
<tr>
<td>planemode</td>
<td>on/off/toggle</td>
</tr>
<tr>
<td>doze</td>
<td>on/off/toggle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special</th>
<th>Param1</th>
<th>Param2</th>
<th>Param3</th>
<th>Param4</th>
<th>Param5</th>
</tr>
</thead>
<tbody>
<tr>
<td>reinit</td>
<td>package</td>
<td>activity</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Appendix C: Automatic Generation of GUI Models Example

**Action:** reinitialize `com.tum.yahtzee MainActivity`
Action: click 200 390 (click play)
Appendix C: Automatic Generation of GUI Models Example

Action: click 200 410 (click ok)
Appendix C: Automatic Generation of GUI Models Example

**Action:** text 200 270 12345 (text1)
Appendix C: Automatic Generation of GUI Models Example

**Action:** reinitialize com.tum.yahtzee MainActivity

![Diagram of reinitialization process]
**Action:** text 200 270 12345 (text1)
**Action:** text 200 330 12345 (text2)
Appendix C: Automatic Generation of GUI Models Example

**Action:** click 200 390 (click play)
Appendix C: Automatic Generation of GUI Models Example

**Action:** click 200 390 (click play)
Appendix D: Abstraction Functions in the Paper

\[ \beta(v) = \begin{cases} 
1, & |\lambda(v)| \leq 1 \\
2, & |\lambda(v)| \leq 3 \\
3, & |\lambda(v)| \leq 8 \\
4, & |\lambda(v)| \leq 15 \\
5, & |\lambda(v)| > 15 
\end{cases} \]

\[ \alpha(z) = \begin{cases} 
1, & z \text{ is a } menu \\
2, & z \text{ is a } back \\
3, & z \text{ is a } click \\
4, & z \text{ is a } longclick \\
5, & z \text{ is a } text \\
6, & z \text{ is a } swipe \\
7, & z \text{ is a } contextual 
\end{cases} \]

- \( \lambda(v) \) denotes the set of enabled actions in the state \( v \).
- \( \beta(v) \) and \( \alpha(z) \) abstract states and actions, respectively.
- These abstraction functions are simple and arbitrary. They are open to improvement.
Appendix E: Benchmark Characteristics

Figure: Characteristics of Training and Test Sets

Between

- 0.01-25 MB, 1000-250000 instructions, and 10-20000 methods